Wrocław University of Science and Technology

Metaheuristic Algorithms

Analysis of Hybrid Approaches on
Simulated Annealing to solve the Traveling Salesman Problem

Marcel Jerzyk
May 31, 2020

Contents

Abstract

The Traveling Salesman Problem (*TSP*) is one of the most frequently discussed NP-Complete [1] optimization problem. The processing time needed to solve it by an exhaustive method to assure the best solution has exponential growth. To overcome that, the problem is being heavily studied and investigated to find better, less expensive heuristics. More efficient methodologies are developed due to the prevalence of the problem in everyday applications (like Transportation and Logistics, Vehicle Routing [2,3], Assignment of Routes for Planes, Image Processing and Pattern Recognition [4], Data-Associations, Data Transmissions in Computer Networks [5]). That great interest in efficient heuristic for *TSP* led to a lot of papers with various proposed solutions, among others: The Simulated Annealing [6–8]. The main problem with Simulated Annealing technique is that the iteration process is slow [9,10] and so hybrid algorithms are being tested to improve both speed (time to cost) and quality of the answer. In this paper closer look is taken on Simulated Annealing combined with Tabu Search [9], Greedy Search [10], and Roulette Wheel [11].

KEYWORDS
Traveling Salesman Problem, *TSP*, Simulated Annealing, Tabu Search, Greedy Search, Roulette Wheel, Comparison

1. Introduction

The Traveling Salesman Problem (*TSP*) is deceptively simple and yet it remains a very challenging task in the optimization field. In short, the task is to find the minimum Hamilton cycle (a cycle in which each vertex of the graph is visited exactly once) in a fully weighted graph. The name comes from the most popular representation of the problem in which a travelling salesman has to visit n cities and his goal is to find the fastest, cheapest or shortest road that connects all these cities starting (and/or ending) at specific point.

There has been several algorithms proposed to solve the problem and provide the global optimum solution [13,15,16] and other, heuristic ones that do not guarantee the optimal solution but are much faster.

2. Travelling Salesman Problem

TSP is well-known and widely studied combinatorial problem in operations research, however there exists no exact method to find optimal *TSP* solution in polynomial time. Thus, it is reasonable to find near optimal solutions within "acceptable" times.

The fundamental formula for travelling salesman problem is given below [12]:

$$Min. \sum_{i=1}^{n} \sum_{i \neq j, j=1}^{n} c_{ij}, x_{ij}$$

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1, \qquad (j = 1, .., n)$$

$$\sum_{j=1, i \neq j}^{n} x_{ij} = 1, \qquad (i = 1, .., n)$$

$$u_i - u_j + n x_{ij} \leq n - 1, \qquad 2 <= i \neq j \leq n$$

$$x_{ij} \in \{0, 1\}$$

Where in the formula n is the number of cities, c_{ij} is the cost of travelling ebtween city i and city j, x_{ij} is a decision variable (like a boolean - it is equals to 1 wherever a connection between city i and city j occurs). For symmetrical *TSP*, the number of possible solutions for n cities is $\frac{(n-1)!}{2}$.

TSPLib[1] library provides test data sets on that problem which are widely used among papers and articles about this problem.

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ (May 31, 2020)

3. Simulated Annealing

Simulated Annealing (*SA*) [17] is one of the probabilistic meta-algorithm for the global optimization problem (locating a good approximation to the global minimum of a function in a large search space). The name come from physical process of annealing - where the metal is the most vivid while the hottest and with the decrease in temperature it is less an less plastic. With that cooling schedule the behavior of *SA* can be controlled. Another property of *SA* is that it is not dependent on the initial solution. With these two adventages comes two problems: the initial temperature is difficult to determine and generating optimal or near optimal solution requires large amount of iterations. This is the reason why *SA* is often used in a hybrid manner.

3.1. Tabu Search

Tabu Search (TS) [18] is regarded as a higher-level heuristic approach for solving optimization problems. The main benefit of it is the ability to escape from the local optima during the search by list of prohibited moves which is called *Tabu-List*. The drawback is that the quality of solution is highly dependent on the initial solution and possible cycling - fallowing the same path unless a tabu neighbor exists.

3.2. Greedy Search

Greedy Search (GS) is quite loose term as it stand for any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum. As for *TSP* an example could be visiting the nearest unvisited city at each step of the journey. It does not provide a best solution but it terminates in a reasonable number of steps.

3.3. Roulette Wheel

Roulette Wheel (RW) also known as Fitness Proportionate Selection is a genetic operator used in genetic algorithms for selection of potentially useful solutions. Fitness function assigns a fitness to possible solutions. This fitness level is used to associate a probability of selecting each individual - candidates with higher fitness will be less likely to be eliminated. In the case of *TSP* since the goal is to minimize the total length of route, shorter edges will enter the solution with higher probability.

4. Methods

The conventional *SA* is rather simple and yet effective. It uses neighborhood search. The possibility of accepting worse solution in *TSP* is controlled by the fallowing formula:

$$p(x^* = x_2) = \begin{cases} e^{\frac{Z_1 - X_2}{T}}, & \text{if } Z_2 \geq Z_1. \\ 1, & \text{if } Z_2 < Z_1. \end{cases} \tag{1}$$

where Z_1 and Z_2 are objective values of x_1 and x_2 solutions and T is the Tem-

perature parameter of *SA*. At first, *SA* starts with the initial solution x and initial temperature T. Temperature is chosen relatively high at initial state. With each step of the algorithm, the temperature decreases by some cooling constant, that is usually set in range $0.7 - 0.99$ [14].

Summing it all briefly, *SA* minimizes $F(c) = \sum_{i=1}^{n-1} L_{c_i,c_j} + L_{c_1,c_n}$ by utilizing cooling $T_{k+1} = \alpha * T_k$ on temperature T that works as a probability check to accept worse but new solution in order to escape local minima with goal to reach global minimum. Here's overview of the algorithm:

- Create random initial solution and set initial temperature
- Begin looping until stop condition is met, either:
  - good-enough has been found
  - system has sufficiently cooled down
- Select a neighbor by making a small change to current solution
- Decide whether to move to the neighbor solution
- Decrease the temperature and go back to the beginning of the loop

4.1. Tabu-Simulated Annealing

SA with its stochastic characteristics takes care of the drawback of TS which is cycling due to its the deterministic nature. *SA* has no memory of recently visited solutions so the rate of improvement is very slow - there's always a probability that search will return to the same solution and that's assisted by the TS *Tabu-List* by restricting previously visited solutions. That can significantly enhance the performance of *SA*.

4.1.1. Cooling Schedule

This approach addresses the problem of reduced change of uphill transition as the temperature continuously declines. To alleviate that, there's proposed adaptive simulated annealing cooling that adjusts the temperature dynamically based on the profile of the search path. Those adjustments make the function not only monotonically non increasing but even allows reheating.

The temperature is controlled by a single function that maintains the temperature above minimum level. If path is going uphill then the temperature will gradually increase. The cooling is sudden with the first downhill move.

$$t_i = t_{min} + \lambda * ln(1 + r_i)$$

where t_{min} is the minimum value that the temperature can take, λ is coefficient that controls the rate of temperature rise and r_i is the number of consecutive upward moves at iteration i The initial value of r_i is zero. That prevents the cool/heat function from becoming invalid when r_i is zero and also it determines the initial value of the temperature.

When a large value is assigned to λ the search spends less time looking for good solutions in the current neighborhood - otherwise with small value, it does the opposite. Tuning that value could be also linked to computation time which also depends on the size and complexity of the problem [19]. Of course, this adaptivity is a great advantage - less pre-search analysis is required, as for conventional *SA* such shortcoming could

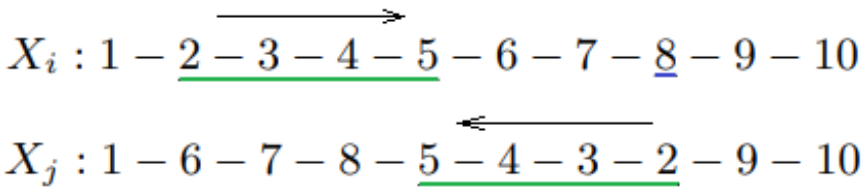


Figure 1. Visual representation of the M_6

also be alleviated but with proper initial temperature and cooling rate.

4.1.2. Neighborhood Generation

In this proposed algorithm neighborhood generation is based on sampling method of reversing and moving sub-permutation as it has highest efficiency for solving [20]. This perturbation method is denoted as M_6 and if X_i denotes the current path of *TSP* problem and $X_J = N(X_i$ denotes the neighbor solution generated by a small perturbation, the M_6 perturbation method is presented in *Figure. 1.* There's also M_1 perturbation which swaps two neighbours with each other.

4.1.3. Tabu-List

At each step of algorithm *Tabu-List* checks whether the solution generated is latterly visited or not and thus help in restricting the algorithm to revisit the pre-visited solutions. This feature of Tabu-Simulated Annealing reduces the computational time of algorithm to reach the global optima.

4.1.4. Parameters

Paper bench-marked the algorithm on M_6 method set for the 75% of the running time and the M_1 method for the rest of time search. Initial Temperature was set to 1 (due to how this method works) and the tabu tenure to the same as in [21]: $n * 2.375$ (where n is the number of cities).

The M_6 has a good chance to direct the searching process into new area, while M_1 neighbourhood perturbation method's relative little impact on sub-path of near optimal path is favorable to the local search stage of the whole searching procedure.

4.2. Greedy Simulated Annealing

In this example *SA* is proposed to be paired with three effective mutations strategy in the local search - Vertex Insert Mutation (VI), Block Insert Mutation (BI) and Block Reverse Mutation (BR). They all have their own universal adaptive parameters to control and achieve better result with a faster convergence depending on the *TSP* task. Again - this approach is invented to achieve a reasonable trade-off between computation time and solution quality.

Table 1. Greedy Simulated Annealing Suggested Parameters

	Initial Value	Meaning
N	-	The number of the cities
OPT	-	The optimal tour length
$t_{initial}$	1000.0	The initial temperature
t_{cool}	$(\alpha * N^{0.5} - 1.0)/(\alpha * N^{0.5})$	The cool coefficient of the temperature
t_{greedy}	$(\beta * N)$	The times of greedy search
t_{end}	0.005 or 0.0025	The end temperature
$t_{current}$	$t_{current} * t_{cool}$	The temperature of the current state
t_v	$(N/10)$	The times of compulsive accept

4.2.1. Mutations & Ratios

The Mutations are described in the 2. The biggest proposed probability has the mutation BR while BI the smallest - the ratios are 89%, 1%, 10% for BR, BI, VI respectively.

4.2.2. Concept & Parameters

The Concept of the greedy search is that the old solution c^{old} is replaced by new neighbor solution c^{new_1} when the formula (of *TSP* problem): $F(c^{new_1}) < F(c^{\text{old}})$ and then turn to the next step. Otherwise the next new neighbour solution c^{new_2} is generated and the old solution c^{old} is replaced by the new neighbour solution c^{new_2} when $F(c^{new_2}) < F(c^{old})$, and then turn to the next step. $c^{new_{t\text{greedy}}}$. Finally, the old solution c^{old} is replaced by the best new solution c^{best} among the t_{greedy} neighbor solutions with probability p:

$$p = e^{-(F(c^{\text{best}}) - F(c^{\text{old}})/t_{\text{current}}) * (10.0 * N/OPT)}$$

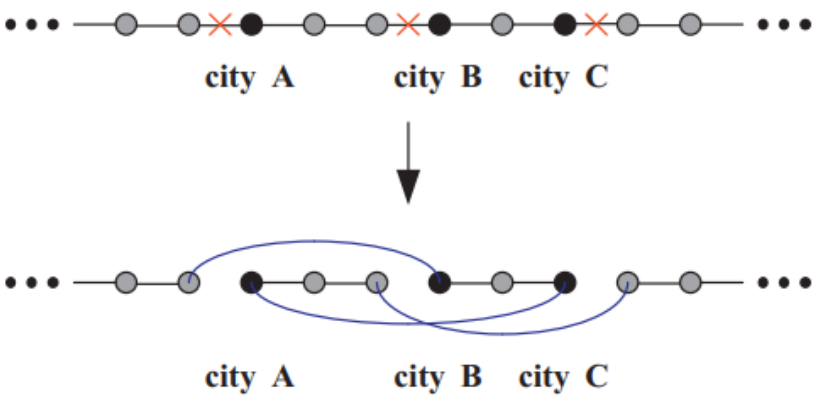
Parameter $t_{current}$ denote temperature of current state, N the number of cities and OPT the optimal tour length.

Having this many variables (t_{cool} (coefficient of the temperature), t_{greedy} (greedy search), t_v (compulsive accept) and p (probability)) makes it difficult and challenging to scale this approach to different scale *TSP* instance.

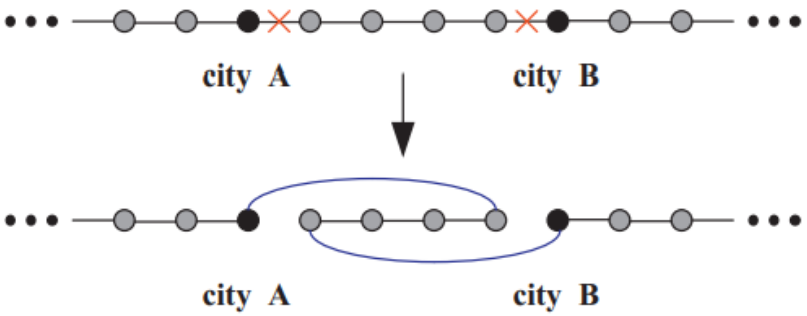
It is suggested that quick cool coefficient is a good fit for small-scale *TSP* instances and a hot initial temperature is a good fit for the *TSP* instances with big optimal tour length. Manipulating t_v is useful to avoid local minima and ultimately finding the desired solution and finally p can be adjusted according to the optimal tour length. The suggested variables in the algorithm are adaptive for cities scale from 51 to 85900 and are shown in the *Table 1*.

4.3. Roulette Simulated Annealing

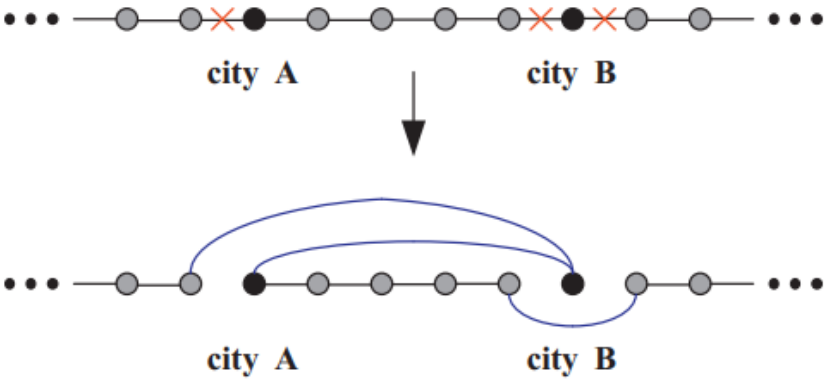
This approach relies on pre-generating data on previous search experience to use in an intelligent and stochastic way for the generation of new solutions by employing Roulette Wheel selection. It does not sacrifice stochasticity and randomness while using problem data which is essential for successful search of solution space.



(a) Block Insert Mutation



(b) Block Reverse Mutation



(c) Vertex Insert Mutation

Figure 2. Visual representation of the three mutations

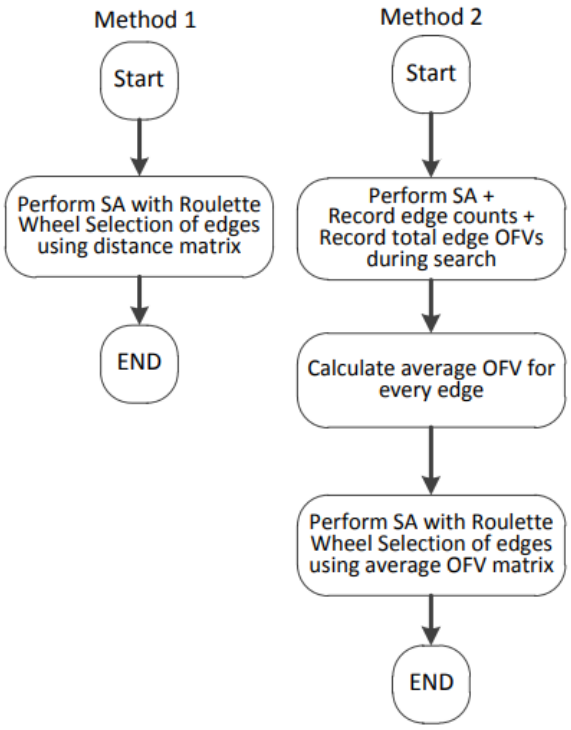


Figure 3. Two Methods of Roulette Simulated Annealing. OFV - objective function values

4.3.1. Two Methods for Getting Fitness

Roulette Simulated Annealing is proposed with two methods: in brief the first is similar to conventional *SA* but the neighbor solutions will be generated with the help of Roulette Wheel selection (shorter paths are selected with higher probability).

In the second method there are two stages - first stage uses conventional *SA* to record the number of times each path is part of solution and objective function values of current solution and neighbour solution. Then the results in both matrices are divided element by element and the average values are used in stage 2, which fallows just like in the first method with the difference that Roulette Wheel selection instead of shorter paths, uses previously calculated values instead. *Figure 3.* shows simple flow diagrams of these two methods.

5. Results

All of these three hybrid *SA* algorithms has proved themselves on bench-marks to outperform conventional *SA* algorithm in a given set of *TSP* problems from the standard library.

5.1. Greedy Simulated Annealing

The effectiveness of the *GSA* for *TSP* was measured with 60 benchmark instances with five trials and the results are in *Table 4* Worth noting is that for *TSP* instances

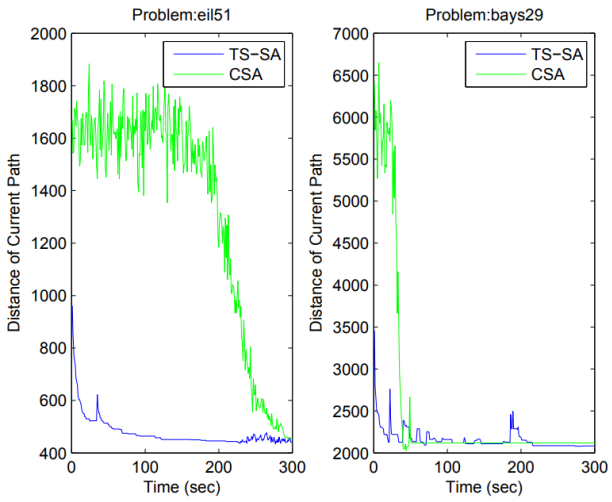


Figure 4. Comparison of convergence speed of Tabu-Simulated Annealing

$KroB100$, $Pr107$ and $Pr144$ best known solutions were replaced with now new best known solutions.

5.2. Tabu-Simulated Annealing

It can been seen in *Figure 4.* that *TSA* algorithm has faster convergence rate compared to conventional *SA* algorithm, especially on the case of $Eil51$ which is a larger problem. Worth noticing is that the *TSA* has better chance of upward move that causes convergence curve progress more turbulent, avoiding the freeze state of conventional *SA*.

5.3. Roulette Simulated Annealing

For testing, 11 example problems from the literature were used in two stages - first one without time limit and second with time limitation. Results are gathered and presented in *Table 3.*

In the first stage the results using first and second method are better compared to the results reached using conventional *SA* with swap operator, but the running time was 18 times longer than in conventional *SA*. That's due to computational complexity. However, when in the second stage when conventional *SA*'s epoch length was increased so that its running time slightly exceeds that of first and second method of Roulette Simulated Annealing, the effectiveness in a limited time is superior in *RSA* compared to the results obtained using conventional *SA*.

6. Comparisons

Thanks to the fact that the *TSP* problem papers are usually being tested on the *TSPLib* we can compare results of these three unique approaches on the same data set (*Table 4*). Though, keep in mind, that their results shouldn't be compared against

Table 2. *GSA* Results on 60 *TSP* Benchmarks from *TSP*Lib

	Instances	Scale	OPT	Best	Average	Worst	Error (%)	CPU time (s)
1	eil51	51	426	428.872	428.872	428.872	0.67	3.91
2	berlin52	52	7542	7544.37	7544.37	7544.37	0.03	3.83
3	st70	70	675	677.11	677.11	677.11	0.31	5.15
4	eil76	76	538	544.369	544.369	544.369	1.18	5.50
5	pr76	76	108159	108159	108159	108159	0.00	5.49
6	rat99	99	1211	1219.24	1219.49	1219.86	0.70	7.34
7	rd100	100	7910	7910.4	7910.4	7910.4	0.00	7.13
8	kroA100	100	21282	21285.4	21285.4	21285.4	0.01	7.14
9	kroB100	100	22141	22139.1	22139.1	22139.1	–	7.14
10	kroC100	100	20749	20750.8	20750.8	20750.8	0.00	7.14
11	kroD100	100	21294	21294.3	21301.0	21311.2	0.03	7.13
12	kroE100	100	22068	22106.3	22112.3	22123.7	0.20	7.14
13	eil101	101	629	640.212	640.515	640.926	1.83	7.42
14	lin105	105	14379	14383	14383	14383	0.02	7.68
15	pr107	107	44303	44301.7	44301.7	44301.7	–	7.78
16	pr124	124	59030	59030.7	59030.7	59030.7	0.00	9.01
17	bier127	127	118282	118294	118349	118418	0.05	9.20
18	ch130	130	6110	6110.72	6121.15	6158.57	0.18	8.00
19	pr136	136	96772	96966.3	97078.9	97186.0	0.31	9.86
20	pr144	144	58537	58535.2	58545.6	58587.1	0.01	10.28
21	ch150	150	6528	6530.9	6539.8	6553.1	0.16	10.91
22	kroA150	150	26524	26524.9	26538.6	26564.2	0.05	10.90
23	kroB150	150	26130	26140.7	26178.1	26223.2	0.18	10.90
24	pr152	152	73682	73683.6	73694.7	73739.1	0.01	10.85
25	u159	159	42080	42392.9	42398.9	42408	0.75	11.49
26	rat195	195	2323	2345.22	2348.05	2352.38	1.07	14.37
27	d198	198	15780	15830.6	15845.4	15877.8	0.41	14.60
28	kroA200	200	29368	29411.5	29438.4	29461.6	0.23	14.26
29	kroB200	200	29437	29504.2	29513.1	29520.8	0.25	14.24
30	ts225	225	126643	126646	126646	126646	0.00	16.05
31	pr226	226	80369	80542.1	80687.4	80808.3	0.39	16.70
32	gil262	262	2378	2393.64	2398.61	2404.96	0.86	19.43
33	pr264	264	49135	49135	49138.9	49142.4	0.00	19.09
34	pr299	299	48191	48269.2	48326.4	48351.4	0.28	21.94
35	lin318	318	42029	42306.7	42383.7	42563.8	0.84	23.35
36	rd400	400	15281	15350.7	15429.8	15498.0	0.97	30.40
37	fl417	417	11861	11940.4	12043.8	12121.0	1.54	32.02
38	pr439	439	107217	110020	110226	110474	2.80	34.92
39	pcb442	442	50778	51063.9	51269.2	51365.0	0.96	35.75
40	u574	574	36905	37232.3	37369.8	37465.3	1.25	48.47
41	rat575	575	6773	6872.11	6904.82	6923.47	1.94	52.10
42	u724	724	41910	42274.7	42470.4	42803.0	1.33	66.83
43	rat783	783	8806	8954.36	8982.19	9013.86	2.00	78.90
44	pr1002	1002	259045	263512	264274	264922	2.01	164.42
45	pcb1173	1173	56892	57760.6	57820.5	57932.2	1.63	193.08
46	d1291	1291	50801	51751.2	52252.3	52798.9	2.85	214.64
47	rl1323	1323	270199	271964	273444	275315	1.20	210.16
48	fl1400	1400	20127	20647.4	20782.2	20956.8	3.25	232.02
49	d1655	1655	62128	63635.9	64155.9	64509.9	3.26	281.88
50	vm1748	1748	336556	342437	343911	344812	2.18	276.98
51	u2319	2319	234256	236356	236744	237077	1.06	410.97
52	pcb3038	3038	137694	140742	141242	142062	2.57	554.28
53	fnl4461	4461	182566	186956	187409	187928	2.65	830.90
54	rl5934	5934	556045	572380	575437	580017	3.48	1043.95
55	pla7397	7397	23260728	24101920	24166453	24195636	3.89	1245.22
56	usa13509	13509	19982859	20799629	20811106	20827805	4.14	2016.05
57	brd14051	14051	469385	485622	486762	486197	3.58	2080.50
58	d18512	18512	645238	668104	669445	670812	3.75	2593.97
59	pla33810	33810	66048945	69391254	69533166	69693399	5.27	4199.88
60	pla85900	85900	142382641	155290611	156083025	156631417	9.62	8855.13

Table 3. First and Second Stage of the Experiment

	Tmax	Tmin	Epoch	Cooling	Min	Max	Avg	Std. dev.	of Sol. Searched	Time (avg.)	Avg. Err	Optimal
Method 1												
Gr17	100000	1	20	0.99	2085	2149	2092.5	14.73	22900	0.69	0.36	2085
Gr21	100000	1	20	0.99	2707	3000	2738.6	72.69	22900	0.75	1.17	2707
Gr24	100000	1	20	0.99	1272	1408	1322.7	37.65	22900	0.75	3.99	1272
Bayg29	100000	1	20	0.99	1610	1813	1658.2	53.50	22900	0.80	2.99	1610
Dantzig42	100000	1	20	0.999	699	740	713.5	12.99	230140	9.69	2.08	699
Gr48	100000	1	20	0.999	5072	5414	5195.2	95.56	230140	10.14	2.96	5046
Berlin52	100000	1	20	0.999	7542	8719	8049.2	281.24	230140	11.36	6.73	7542
St70	100000	1	20	0.999	678	797	727.9	26.29	230140	13.69	7.84	675
Pr76	100000	1	20	0.999	109769	125739	117335.2	4068.60	230140	15.93	8.48	108159
KroA100	100000	1	200	0.999	21343	23567	22129.1	573.04	2301400	212.90	3.98	21282
KroA150	100000	1	200	0.999	27176	30549	28494.4	782.76	2301400	388.93	7.43	26524
Method 2												
Gr17	100000	1	20	0.99	2085	2103	2089.7	7.05	22900	0.70	0.23	2085
Gr21	100000	1	20	0.99	2707	3071	2737.0	85.51	22900	0.73	1.11	2707
Gr24	100000	1	20	0.99	1279	1382	1318.0	28.15	22900	0.74	3.61	1272
Bayg29	100000	1	20	0.99	1610	1772	1652.7	44.66	22900	0.77	2.65	1610
Dantzig42	100000	1	20	0.999	699	753	713.7	15.02	230140	9.87	2.10	699
Gr48	100000	1	20	0.999	5055	5326	5144.4	66.98	230140	10.50	1.95	5046
Berlin52	100000	1	20	0.999	7542	8261	7948.9	172.86	230140	11.57	5.40	7542
St70	100000	1	20	0.999	684	777	718.9	23.13	230140	14.10	6.50	675
Pr76	100000	1	20	0.999	110458	124109	114672.7	2505.19	230140	15.19	6.02	108159
KroA100	100000	1	200	0.999	21282	23620	21831.9	536.10	2301400	213.03	2.58	21282
KroA150	100000	1	200	0.999	27315	28788	28002	426.79	2301400	387.26	5.57	26524
***SA* w/ Swap #1**												
Gr17	100000	1	20	0.99	2085	2210	2129.7	36.53	22900	0.24	2.14	2085
Gr21	100000	1	20	0.99	2707	3276	2922.8	197.42	22900	0.26	7.97	2707
Gr24	100000	1	20	0.99	1272	1476	1393.6	42.15	22900	0.24	9.56	1272
Bayg29	100000	1	20	0.99	1653	1979	1805.2	81.88	22900	0.23	12.12	1610
Dantzig42	100000	1	20	0.999	710	879	783.0	40.54	230140	2.37	12.02	699
Gr48	100000	1	20	0.999	5054	6361	5674.5	261.67	230140	2.24	12.46	5046
Berlin52	100000	1	20	0.999	7872	9249	8469.2	301.10	230140	2.356	12.29	7542
St70	100000	1	20	0.999	782	957	868.6	37.98	230140	2.20	28.68	675
Pr76	100000	1	20	0.999	122859	147863	136155.8	6023.71	230140	2.43	25.88	108159
KroA100	100000	1	200	0.999	23677	27760	25597.7	1157.55	2301400	21.89	20.28	21282
KroA150	100000	1	200	0.999	31744	37599	34832.7	1315.70	2301400	21.67	31.33	26524
***SA* w/ Swap #2**												
Gr17	100000	1	60	0.99	2085	2136	2104.9	17.65	68700	0.70	0.95	2085
Gr21	100000	1	60	0.99	2707	3254	2828.5	159.06	68700	0.76	4.49	2707
Gr24	100000	1	65	0.99	1272	1432	1340.9	47.10	74425	0.76	5.42	1272
Bayg29	100000	1	70	0.99	1620	1808	1700.5	57.55	80150	0.81	5.62	1610
Dantzig42	100000	1	85	0.999	699	784	741.6	23.48	978095	10.09	6.09	699
Gr48	100000	1	95	0.999	5071	5709	5373.3	202.66	1093165	11.03	6.49	5046
Berlin52	100000	1	99	0.999	7670	8613	8194.8	248.10	1139193	12.52	8.66	7542
St70	100000	1	129	0.999	704	813	757.5	27.48	1484403	14.20	12.22	675
Pr76	100000	1	150	0.999	113943	132597	122147.2	4150.12	1726050	16.03	12.93	108159
KroA100	100000	1	2050	0.999	21715	24693	23295.8	738.29	23589350	235.94	9.46	21282
KroA150	100000	1	3650	0.999	27869	30931	29541.6	871.87	41296044	395.33	11.38	26524

Table 4. *GSA*, *TSA*, *RSA*

	Instances	Scale	OPT	Best	Average	Worst	Error (%)	CPU time (s)
***GSA* (2011)**								
	St70	70	675	677.11	677.11	677.11	0.31	5.15
	Eil76	76	538	544.369	544.369	544.369	1.18	5.50
	Pr76	76	108159	108159	108159	108159	0.00	5.49
	KroA150	150	26524	26524.9	26538.6	26564.2	0.05	10.90
	KroB150	150	26130	26140.7	26178.1	26223.2	0.18	10.90
***TSA* (2009)**								
	Gr48	48	5058	5125	5295.5	-	-	-
	Berlin52	52	7542	7648	7718.5	-	-	-
	Eil76	76	538	557	564.0	-	-	-
***RSA* (2013)**								
	Gr48	48	5046	5055	5195.2	5414	1.95	10.50
	Berlin52	52	7542	7542	8049.2	8719	11.36	6.73
	St70	70	675	684	718.9	777	6.50	14.10
	Pr76	76	108159	110458	114672.7	124109	6.02	15.19
	KroA100	100	21282	21282	21831.9	23620	2.58	213.03
	KroA150	150	26524	27315	28002	28788	5.57	387.26

each other but rather relatively to themselves - due to different pc configuration on which these were run the results can differ because of the available computational power.

Two of these methods - *GSA* and *TSA* focused on improving *SA* by speeding up the convergence rate either by restricting the pool with *Tabu-List* or by utilizing gready search. In *Table 4.* and *4.* we see that both achieved their main planned goals.

Both *TSA* and *RSA* uses space to improve the speed. Altough *TSA Tabu-List* is linear with relationship with the city count (if amount of cities does not exceed 100) so the use of space is really narrow, the *RSA* uses one or two (depending on a method) whole additional matrix/matrices for all the possible moves.

GSA and *TSA* proposes changes to the cooling process. *GSA* uses adaptive temperature based on the constant and the number of cities and *TSA* removes initial temperature from tweak list entirely and uses heating accompanied with cooling to automate it instead.

Only *RSA* has some pre-start setup with it's second method which require to run conventional *SA* once in order to get the number of times an edge was a part of solution and the average OFV value.

7. Conclusions

All of these hybrid approaches recognises the main weakness of *SA* which is the convergence rate. All of them are improving it in order to achieve better time/quality results. Additionally two of them uses space to achieve better results, two of them changes the cooling calculations and two of them and two of them uses some kind of perturbations.

7.1. TSA

7.2. Wrap-up

TSA uses the *Tabu-List* to achieve conspicuously fast convergence rate compared to conventional *SA* and then with its new temperature control scheme tries to escape from local minima with it's higher chance of an uphill move at the end of search. The downside of that is that there's a chance to explore worse solution at the very

end of search which should be taken into consideration when using this method. The unique heating approach additionally reduces the need of investigating proper initial temperature and thus - more simplicity.

7.2.1. Suggestions

- Manipulate mutations (these recommendations are based on example of 783 cities)
  - BR alone obtains best solution with shortest time
  - BI solely obtains worst solution with worst time
  - Together, mutations with proposed ratios (BR, VI, BI) (89%, 10%, 1%) perform better than BR individually
- For small scale TSP instance - use quick cool coefficient
- Increase t_v to avoid local minimas

7.3. GSA

7.4. Wrap-up

GSA also focus on speeding up convergence rate but instead uses greedy search technique. Then with use of several adaptive parameters (for cool coefficient of the temperature, time of greedy search, time of compulsive accept, and the probability of new solution) tries to achieve best quality-time trade off. However, if dealing with very large *TSP* problems the time to reach an answer increases and the time/cost effectiveness is dim compared to some of the genetic algorithm approaches [22,23]. Another con is the implementation which is more complicated to classic and the other two presented options - both in terms of various methods inside the algorithms and amount of variables to tweak.

7.4.1. Suggestions

- increase the λ to make the algorithm spend less time for looking for good solutions in current neighborhood
- decrease the λ to make the algorithm spend more time looking for better solutions in the neighborhood

7.5. RSA

7.6. Wrap-up

RSA on the other hand presents another way of improvement: using pre-computated data with one of the core genetic algorithms based feature - fitness - to make the path picking decision smarter. The main advantage is again - improved convergence capabilities of the algorithm by using problem data without losing inherent stochasticity in the *SA* technique. In comparison with *SA* in a given time frame the quality of final answers were found to be superior.

7.6.1. Suggestions

- For *TSP* with less than 50 cities set cooling rate λ to 0.99
- For *TSP* with more than 50 cities set cooling rate λ to 0.999

Disclosure statement

The author declares that he has no relevant or material financial interests that relate to the research described in this paper.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors

Notes on contributor(s)

Student of Wroclaw University of Science and Technology at the faculty of Fundamental Technical Problems.

References

[1] C.H. Papadimitriou, Euclidean traveling salesman problem is NP-complete, Theoretical Computer Science 4 (1978) 237–244.

[2] Y. Marinakis, M. Marinaki, G. Dounias, A hybrid particle swarm optimization algorithm for the vehicle routing problem, Engineering Applications of Artificial Intelligence 23 (2010) 463–472.

[3] K. Savla, E. Frazzoli, F. Bullo, Traveling salesperson problems for the Dubins vehicle, IEEE Transactions on Automatic Control 53 (6) (2008) 1378–1391

[4] D. Banaszak, G.A. Dale, A.N. Watkins, J.D. Jordan, An optical technique for detecting fatigue cracks in aerospace structures, in: Proc. 18th ICIASF, 1999, pp. 1–7.

[5] M.K. Mehmet Ali, F. Kamoun, Neural networks for shortest tour computation and routing in computer networks, IEEE Transactions on Neural Network 4 (5) (1993) 941–953.

[6] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Configuration space analysis of traveling salesman problem, Journal Physique 46 (1985) 1277–1292.

[7] K. Meer, Simulated annealing versus metropolis for a *TSP* instance, Information Processing Letters 104 (6) (2007) 216–219.

[8] F. Tian, L. Wang, Chaotic simulated annealing with augmented Lagrange for solving combinatorial optimization problems, in: Proc. 26th Annual IECON, vol. 4, 2000, pp. 2722–2725.

[9] Yi Liu, Shengwu Xiong, Hongbing Liu Hybrid simulated annealing algorithm based on adaptive cooling schedule for *TSP*; GEC '09: Proceedings of the first ACM/SIGEVO, Pages 895–898

[10] Xiutang Genga, Zhihua Chenb, Wei Yanga, Deqian Shia, Kai Zhaoa, Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search

[11] Bayram, Hüsamettin Şahin, Ramazan. (2013). A new simulated annealing approach for the traveling salesman problem. Mathematical and Computational Applications. 18. 313-322. 10.3390/mca18030313.
[12] Miller, C.E., Tucker, A.W., Zemlin, R.A.: integer programming formulation of traveling salesman problems. J. ACM 7(4), 326-329 (1960)
[13] T.Guo and Z.Michalewize. Inver-over operator for the *TSP*. In Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, pages 802–812. IEEE, July 1998.
[14] Halim, A.H., Ismail, I.: Combinatorial optimization: comparison of heuristic algorithms in travelling salesman problem. Arch. Comput. Methods Eng. 26(2), 367-380 (2019)
[15] Z.C.Huang, X.L.Hu, and S.D.Chen. Dynamic traveling salesman problem based on evolutionary computation. In Proceedings of Congress on Evolutionary Computation, pages 1283–1288. IEEE, May 2001.
[16] Aimin Zhou, Lishan Kang, and Zhenyu Yan. Solving dynamic *TSP* with evolutionary approach in real time. In Proceedings of Congress on Evolutionary Computation, pages 951–957. IEEE, December 2003.
[17]] C. D. Gelatt Jr S. Kirkpatrick and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983
[18] F. Glover. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13(5):533–549, 1986.
[19] Nader Azizi and Saeed Zolfaghari. Adaptive temperature control for simulated annealng:a comparative study. Computers and Operations Research, 13(31):2439–2451, 2004.
[20] P. Tian, H. Wang, and D. Zhang. Solving the travelling salesman problem by simulated annealing. Journal of Shanghai Jiaotong University, 29(12):111–116, 1995
[21] T. H. Wu and C. C. Chang. A tabu search heuristic for the traveling salesman problem. Journal of Da-Yeh University, 19(1):87–99, 1997.
[22] M. Saadatmand-Tarzjan, M. Khademi, M.R. Akbarzadeh-T, H.A. Mghaddam, A novel constructive-optimizer neural network for the traveling salesman problem, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 37 (4) (2007) 754–770.
[23] J. Yang, C. Wu, H.P. Lee, Y. Liang, Solving traveling salesman problem using generalized chromosome genetic algorithm, Progress in Natural Science 18 (2008) 887–892.